Project 3: Finding minima of a nonlinear function

1 The problem

In many applications both in applied sciences and in industry an important problem is that of finding local or global maxima or minima of nonlinear functions of several variables. The problem of finding a local minimum may be written: given $f : \mathbb{R}^N \mapsto \mathbb{R}$ find $\overrightarrow{x} \in \mathbb{R}^N$ such that, for some $\epsilon_0 > 0$,

$$f(\overrightarrow{x}) \leq f(\overrightarrow{x} + \overrightarrow{y}), \text{ for all } \overrightarrow{y} \in \mathbb{R}^N \text{ such that } |\overrightarrow{y}| \leq \epsilon_0.$$

Such a point is known as a critical point as we have seen in the course. Provided $f(\cdot)$ is smooth enough, it is characterised by the fact that the gradient of f vanishes at $\overrightarrow{x}$, $\overrightarrow{\nabla} f(\overrightarrow{x}) = \overrightarrow{0}$ and that the Hessian evaluated at this point is positive defininite.

The algorithm that we will use to find approximate local minim=a is known as the *steepest descent algorithm* and is an iterative procedure that starts from some initial guess $\overrightarrow{x}_0 \in \mathbb{R}^N$ and then creates a sequence of updates, $\overrightarrow{x}_k$ created by "moving" in the direction in which f decreases the most, the direction of "steepest descent". Given $\overrightarrow{x}_0$, a tolerance `TOL` and some upper limit on the number of iterations, `maxit` we can write the algorithm for minimisation, at iteration k, as follows

1. Compute $\overrightarrow{g}_k = \overrightarrow{\nabla} f(\overrightarrow{x}_k)$.
2. If $|\overrightarrow{g}_k|$ <`TOL` or k >`maxit`: return x_k, $|g_k|$ and k and claim success if $|\overrightarrow{g}_k|$ <`TOL` and failure if k >`maxit`.
3. Consider the one dimensional function $\tilde{f}(s) = f(\overrightarrow{x_k} - s\overrightarrow{g}_k)$.
4. Pick an $s_k > 0$ such that $\tilde{f}(s_k) < \tilde{f}(0)$. Since we know that the gradient points in the direction of largest increase of $\overrightarrow{f}$, we see that $f(\overrightarrow{x_k}) > f(\overrightarrow{x_k} - s\overrightarrow{g}_k)$ for some $s > 0$, small enough.
5. Update the point:
$$\overrightarrow{x}_{k+1} = \overrightarrow{x}_k - s_k \overrightarrow{g}_k$$
6. Increase the iteration counter, k, by one, `k+=1`, and repeat from point 1.

Observe that the choice of s_k is not specified above and left to your imagination. The procedure of finding the minimum of this one dimensional

minimisation problem is known as a"line search". A naive (and expensive) way to find an acceptable s_k is to fix a small value of $\epsilon > 0$, set $n = 1$ and then loop (if we identify $\tilde{f}$ with F).

```
while (F(n*epsilon) <F((n-1)*epsilon)):
  n+=1
s_k = (n-1)*epsilon
```

2 The project

The aim of the project is to write a program that uses the above presented algorithm to find the minimum of a given function.

2.1 Program structure

1. Open a file `filename.opt` for reading.
2. Read the data N, a string containing the function f, a string `gradient` that contains either the N components of the gradient of f or the string `'unknown'`. Then read the initial guess $\overrightarrow{x}_0 \in \mathbb{R}^N$, the parameters `TOL` and finally `maxit`.
3. An example of the file (`quadratic.opt`) reads:

   ```
   2
   x[0]**2-1.5*x[0]*x[1]+x[1]**2 - x[0]-x[1]
   2*x[0]-1.5*x[1]-1 2*x[1]-1.5*x[0]-1
   -2.0 3.0
   0.001
   2000
   ```

4. Then use the above algorithm to find a (local) minimum of the function f, starting from the initial guess $\overrightarrow{x}_0$.

2.2 Computation and output

The program should execute the algorithm taking into account the different data given in the following way

- If `gradient != 'unknown'`
  then execute the above algoritm using the provided gradient. Store all the intermediate points x_k in a list.

- If `gradient == 'unknown'`
  execute the above algoritm, but this time approximate the components of the gradient using finite differences:

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(\overrightarrow{x} + \overrightarrow{\epsilon}) - f(\overrightarrow{x} - \overrightarrow{\epsilon})}{2\epsilon}$$

  where $\overrightarrow{\epsilon}$ is a vector with elements $\epsilon_k = 0.$ for $k \neq i$ and $\epsilon_i = 1.e-6$. Store all the intermediate points x_k in a list.

- Return whether or not the tolerance has been met and how many iterations that have been used. Also return the size of the gradient for the last iteration.

- If $N = 2$, produce contourplots of the function f in a square, centered in $\overrightarrow{x}^*$, where $\overrightarrow{x}^*$ is the approximate local minimum, and with side $2|\overrightarrow{x}_0 - \overrightarrow{x}^*|$. Choose the parameters for the plot command so that it is easy to see that there is a local minimum in $\overrightarrow{x}^*$.

3 What to submit for the assessment

You should submit the following items in a zipped folder:

- A .pdf document with details on the members of the group, names and student numbers. Here you may also give some information on the project, such as if you have implemented some particular line search algorithm or some of the extensions below have been considered.

- A well commented code that produces the expected output as outlined in the discussion above.

- The output of the program run using the following three input files (available on the moodle page):

  - `quadratic.opt`
  - `quartic.opt`
  - `quartic_unknow.opt`

- Include both text and graphical output (graphics can be saved using the buttons on the bottom of the Figure window).